# UNIT-5 A8601

**PRESENTED BY**
M.YGANDHAR
Department of IT
Vardhaman College of Engineering

01 TOPIC AWT Hierarchy, AWT controls

02 TOPIC Layout Managers- FlowLayout, BorderLayout

03 TOPIC GridLayout and CardLayout. Limitations of AWT

04 TOPIC Applets- Definition, Life Cycle and Execution

05 TOPIC SWINGS: JFrame, JPanel, JComponent

06 TOPIC JLabel and Image-Icon

07 TOPIC JTextField,JTabbedPane , Swing Buttons

08 TOPIC JScrollPane, JComboBox, JTable.

# Applet

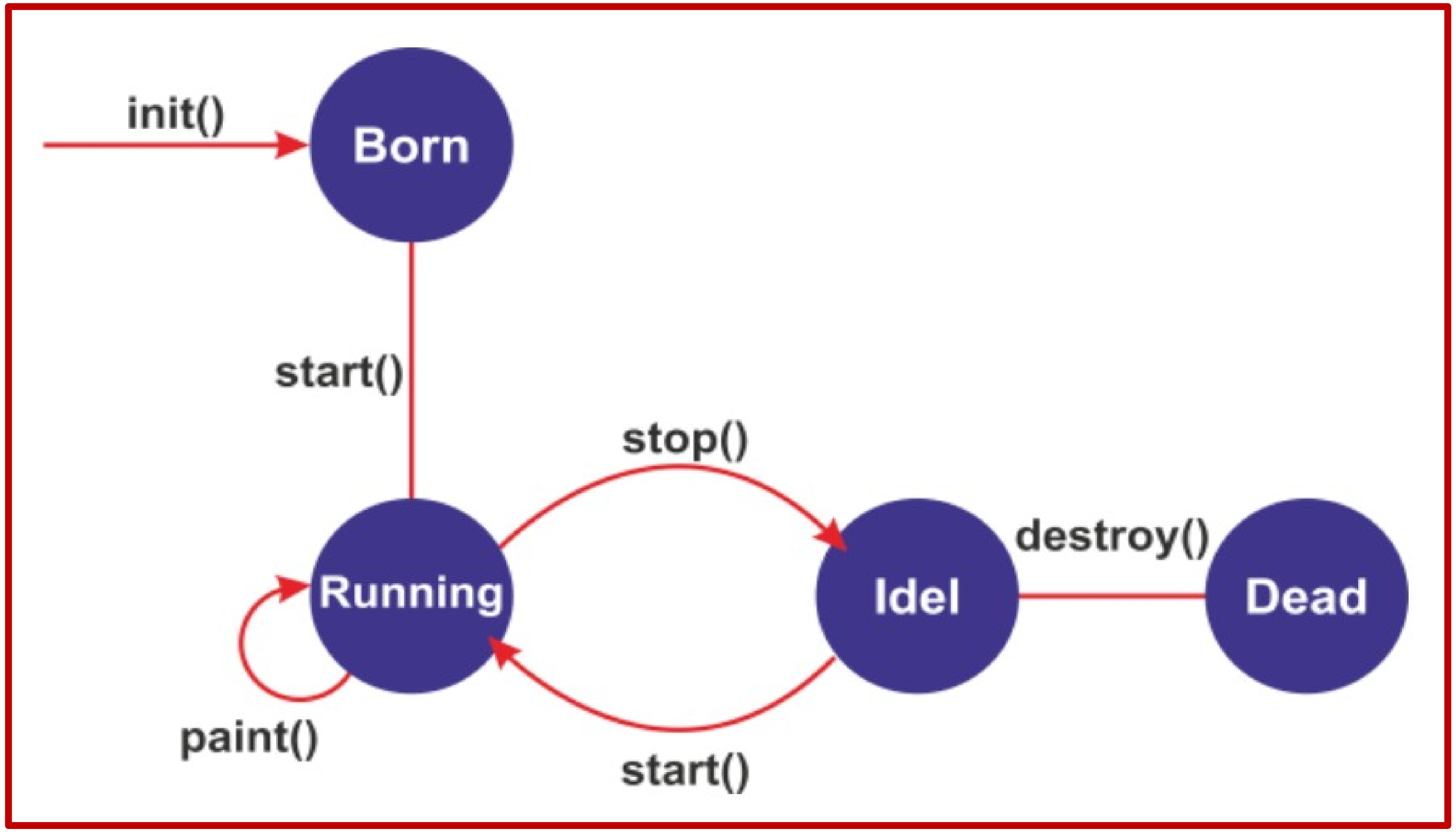
- An applet is a **special kind** of **Java program** that **runs** in a Java **enabled browser.**
- This is the **first Java program** that can **run over the network** using **the browser. Applet is** typically **embedded inside** a **web page** and **runs** in the **browser.**
- To **create an applet**, a class **must** class **extends Applet class**.
- An **Applet class does not** have any **main() method.**
- The **JVM can use** either a plug-in of the **Web browser** or a **separate runtime environment** to **run an applet application**.
- **JVM creates** an **instance** of the **applet class** and **invokes init() method** to **initialize an Applet**.
- **Every Applet application must import two packages** - **java.awt** and **java.applet.**
- **The class** in the **program** must be **declared as public**, because **it will be accessed by code** that is **outside the program**

**How to run an Applet?**

- There are two ways to run an applet

  i. By html file.

# Applet Life Cycle



# Applet Life Cycle

- The **Applet Life Cycle** in Java can be **defined as** the process of **how an applet object is created, started, stopped, and destroyed** during the entire **execution of the applet.**
- There are mainly **five methods used** in the **Applet Life Cycle** in Java namely,

   i. **init()**
   ii. **start()**
   iii. **paint()**
   iv. **stop()**
   v. **destroy()**

**i. init() method:**

✓ It is **used for the initialization** of **the Applet** since no main() method is used.

✓ This init() method is **called only once** for **creating the applet.**

✓ All the **variables are initialized in this method.**

# Applet Life Cycle

## ii. start( ) Method:

- ✓ It is used for **starting the Applet in Java**. This method is **called after the init() method**.
- ✓ This method **contains the actual code** of the **applet.**
- ✓ The **start() method** is **invoked every time** the **browser is loaded** or **refreshed.**

## iii. paint() Method:

- ✓ This step involves **drawing various shapes** in the applet using the paint() method.
- ✓ It **consists** of the **parameter of class Graphics**, which **helps** in enabling the **painting in an applet.**

## iv. stop() method :

- ✓ **To stop an applet, we use the stop()** method of the Applet class.
- ✓ This **stop() method is invoked** when the **browser is minimized**, **restored**, **or moved to another tab.**

## v. destroy() method:

## Applet

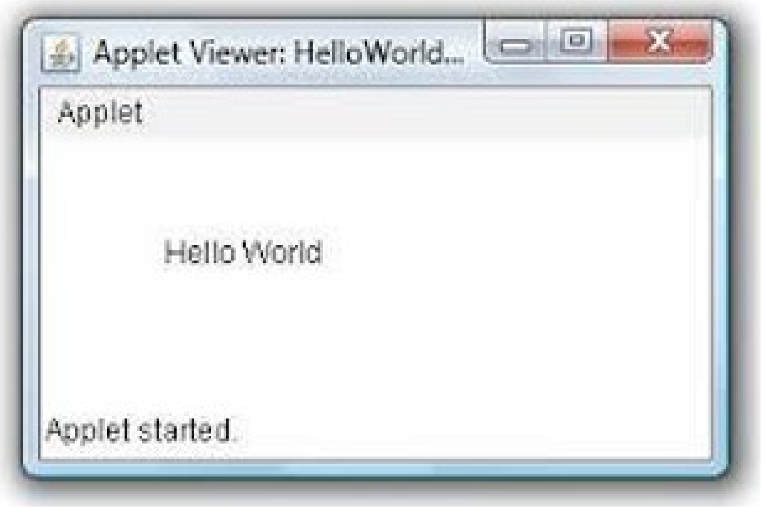
**//Creating applet and run an applet using Java appletviewer**

```
import java.applet.*;
import java.awt.*;
/*
<applet code="First.class" height="100" width="100">
</applet>
*/
public class First extends Applet
{
    public void init()
    {
        setBackground(Color.white);
        setForeground(Color.black);
    }
    public void paint(Graphics g)
    {
        g.drawString(""Hello World", 300, 150);
    }
}
```

**How To Run Applet**

```
c:\>javac First.java
c:\>appletviewer First.java
```

***OUTPUT***



```
Hello World
```

```
Applet started.
```

```
//Creating applet and run the applet using
web browser
import java.applet.*;
import java.awt.*;
public class First extends Applet
{
      public void init()
      {
            setBackground(Color.white);
            setForeground(Color.black);
      }
      public void paint(Graphics g)
      {
            g.drawString(""Hello World", 300, 150);
      }
}
```

```
//Creating html code
<html>
      <body>
      <applet code="First.class" width="150“
height="25"></applet>
      </body>
</html>
```

## How To Run Applet:

**Step-1.**

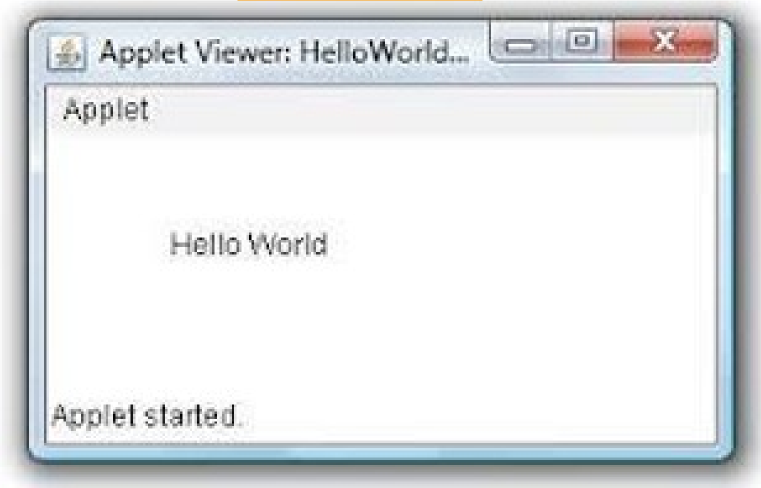
Save html code as "First.html" then compile java code.

**Step-2:**

**c:\>**javac First.java

**Step-3:**

Now double click on First.html file



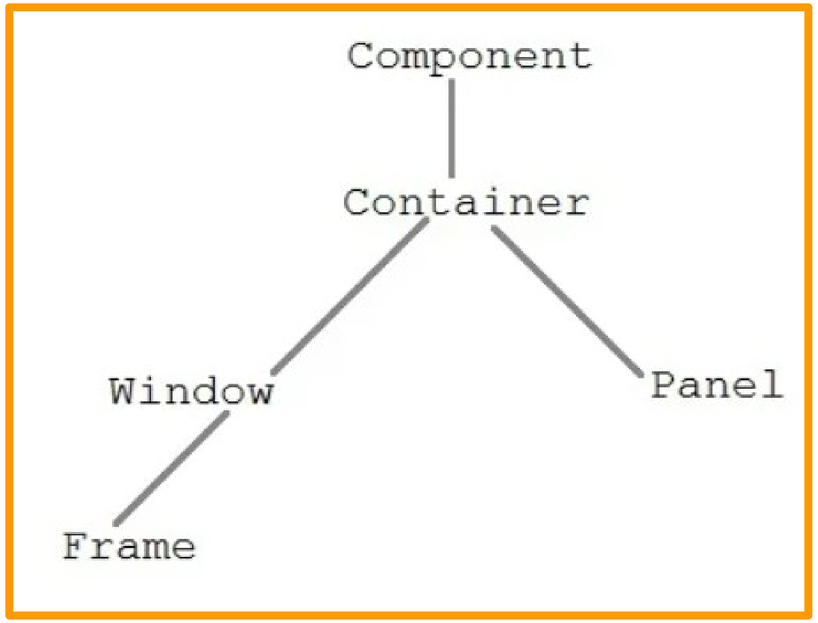
```
Hello World
```

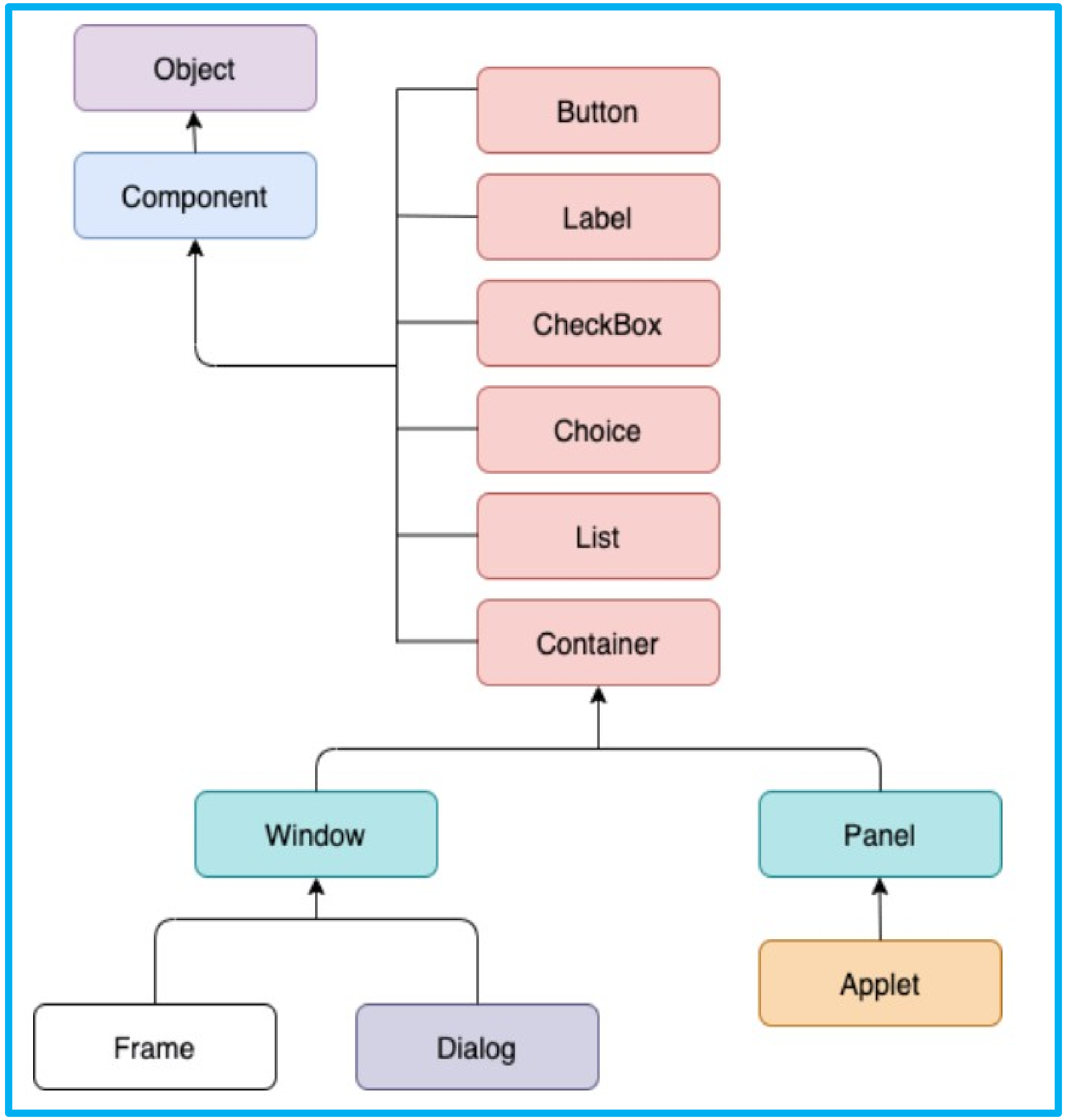
Applet started.

# AWT Hierarchy

- Java **Abstract Window Toolkit (AWT)** is an **API** that **contains large number** of **classes** and **methods** to **create** and **manage graphical user interface ( GUI ) applications.**
- The **AWT** was **designed to provide** a common **set of tools for GUI design** that could **work on a variety** of **platforms.**
- The **tools provided** by **the AWT** are **implemented** using each **platform's native GUI** toolkit, hence preserving the **look and feel** of each platform. This is an advantage of using AWT.
- But the **disadvantage** of such an approach is that **GUI designed on one platform** may **look different** when displayed **on another platform** that means AWT component are platform dependent.
- **AWT is the foundation** upon which Swing is made i.e **Swing is a improved GUI API that extends the AWT.**

# AWT Hierarchy





## AWT Hierarchy

- The **hierarchy of Java AWT classes** are given above diagram, **all the classes** are **available** in **java.awt package.**

**i. Component class:**

- ✓ Component class is at the **top of AWT hierarchy.**
- ✓ **All the elements** like the **button, text fields, scroll bars**, etc. are **called components.**
- ✓ In Java AWT, there are **classes for each component** as shown in above diagram.
- ✓ **In order to place** every **component** in a **particular position** on a **screen**, we need to **add them** to a **container.**

**ii.Container:**

- ✓ **Container** is a component in AWT that **contains** another **component like button, text field, tables** etc.

# Heirarchy of component class

**iii. Window class:**

- ✓ The **window is a container** that **does not have borders** and **menu bars.**
- ✓ In order **to create a window**, you can use **frame or dialog.**

**iv.Frame**

- ✓ Frame is a **subclass of Window** that contain **title bar** and can have **menu bars.**
- ✓ It also **contain several different components** like **button, title bar, textfield, label** etc.
- ✓ **Most of the AWT applications** are created **using Frame window.**

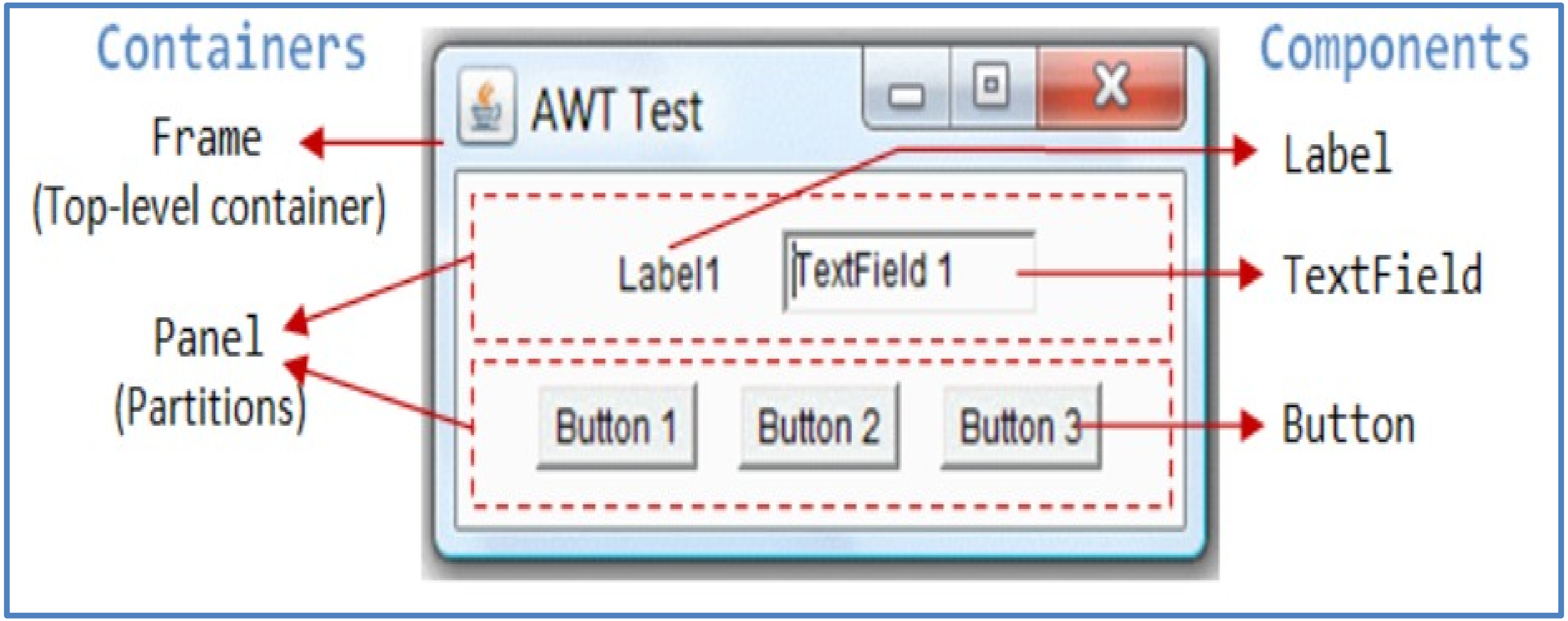
**v.Dialog:**

- ✓ It is the **container having a border and title.**

**vi.Panel:**

- ✓ It is container that is **used for holding components.**
- ✓ [illegible]

# Heirarchy of component class



# AWT controls

**Frame class has two different constructors:**

a. **Frame()**
b. **Frame(String title)**

**Creating a Frame:**

- There are **two ways to create a Frame**. They are,

  i. **By Instantiating Frame class**
  ii. **By extending Frame class**

**Note:**

- **While creating a frame** (either by instantiating or extending Frame class), Following **two attributes are must** for visibility of the frame:

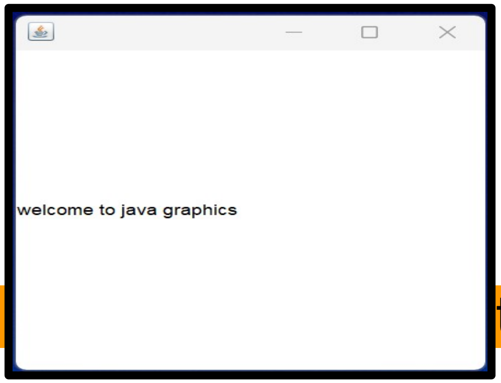
  i. **setSize(int width, int height)**
  ii. **setVisible(true)**

- **When you create other components like Buttons**, TextFields, etc. Then **you need to add it** to the **frame** by using the method :

  **- add(Component's Object)**

# Creating Frame Window by Instantiating Frame class

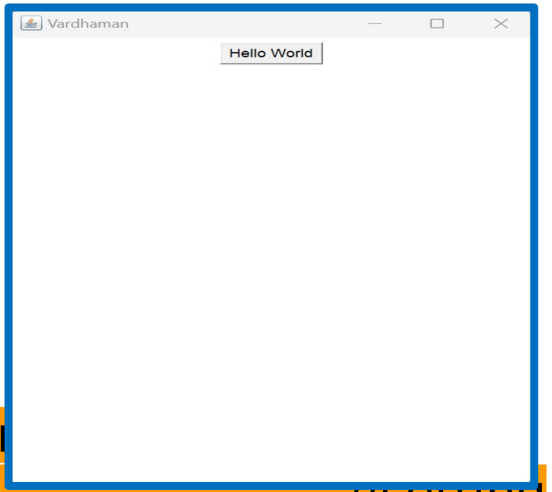
```
//Creating Frame Window by Instantiating Frame class
import java.awt.*;
public class Testawt
{
     Testawt()
     {
          Frame fm=new Frame();                                      ting
a frame
          Label lb = new Label("welcome to java graphics");   //Creating a label
          fm.add(lb);                                                   //adding
label to the frame
          fm.setSize(300, 300);                                        //setting
frame size.
          fm.setVisible(true);                                       //set frame
visibilty true
     }
     public static void main(String args[])
      {
          Testawt ta = new Testawt();
     }
}
```



```
welcome to java graphics
```

# Creating Frame Window by Extending Frame class

```
//Creating Frame window by extending Frame class
package testawt;
import java.awt.*;
import java.awt.event.*;
public class Testawt extends Frame
{
    Testawt()
    {
        Button btn=new Button("Hello World");
        add(btn);                                   //addi
        setSize(400, 500);                                          //setting
size.
        setTitle(“Vardhaman");                                   //setting title.
        setLayout(new FlowLayout());                   //set default layout for
frame.
        setVisible(true);                                           //set frame
visibilty true.
    }
    public static void main (String[] args)
    {
        Testawt ta = new Testawt();
//creating a frame.
    }
}
```

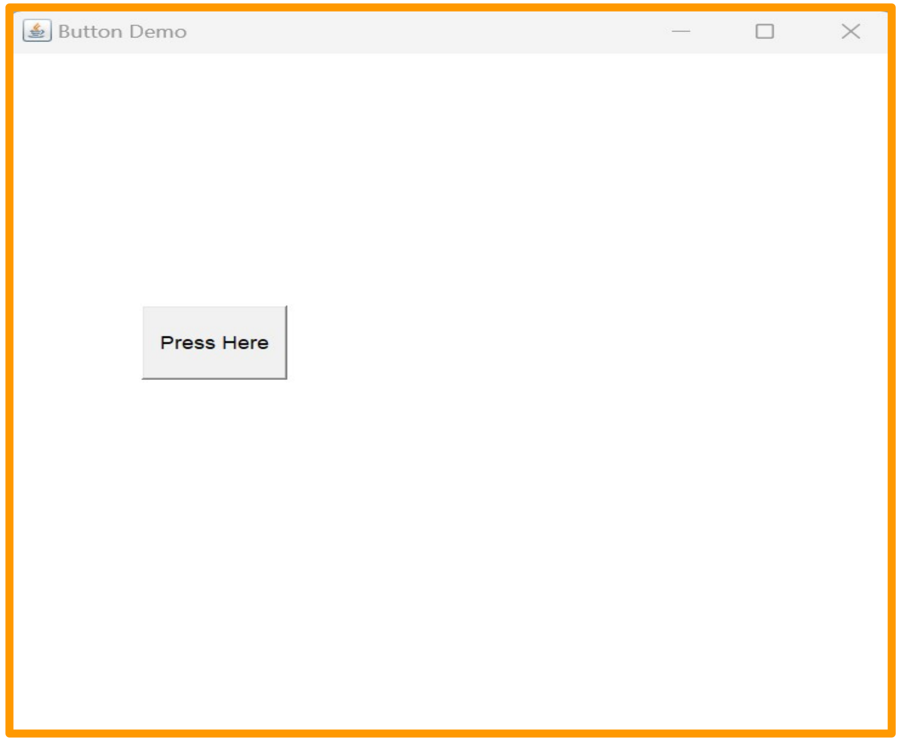


## i. Button Classs

- A **push button** is the **frequently found GUI control.**
- A **button can be created** by using the **Button class** and **its constructors.**
  - ✓ **Button()**
  - ✓ **Button(String str)**
- **Some of the methods available in the Button** class are as follows:
  - ✓ **void setLabel(String str)** – To set or assign the text to be displayed on the button.
  - ✓ **String getLabel()** – To retrieve the text on the button.
- When a **button is clicked**, it **generates an ActionEvent** which can be **handled using the ActionListener** interface and the event handling method is actionPerformed().
- If there are **multiple buttons** we can get the **label of the button which was clicked** by using the method **getActionCommand().**

# Example to create a button

```
import java.awt.*;
public class ButtonDemo1
{
     public static void main(String[] args)
     {
          Frame f=new Frame(“Button Demo");
          Button b1=new Button("Press
Here");
          b1.setBounds(80,200,80,50);
          f.add(b1);
          f.setSize(500,500);
          f.setLayout(null);
          f.setVisible(true);
     }
}
```

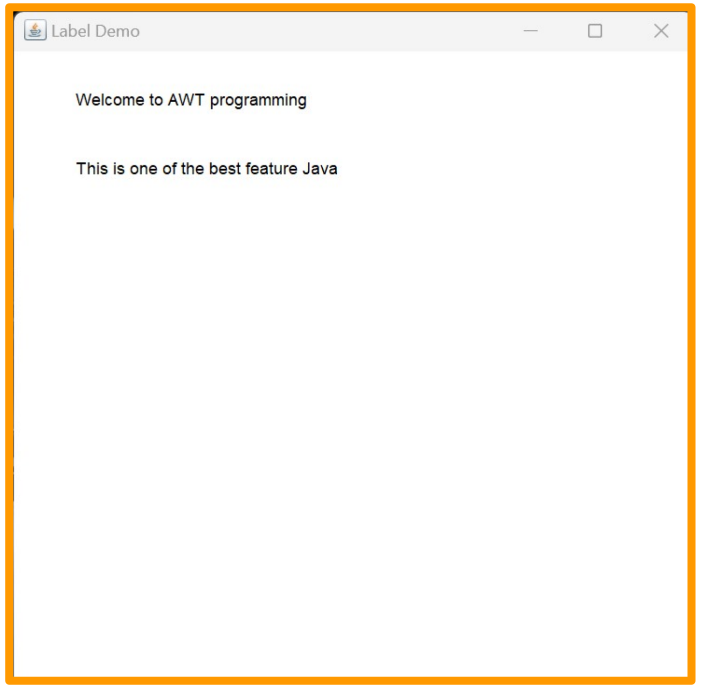


## ii.Label

- It is used for **placing text in a container**. Only **Single line text** is **allowed .**
- A **label** is a GUI control which can be **used to display static text**.
- Label can be **created** using the **Label class** and its **constructors** which are listed below:
  - ✓ Label()
  - ✓ Label(String str)
  - ✓ Label(String str, int how)
- The parameter **how specifies the text alignment.**Valid values are **Label.LEFT, Label.CENTER or Label.RIGHT**
- Some of the **methods available** in the **Label class** are as follows:
  1. void setText(String str) – To **set** or **assign text** to the label.
  2. void setAlignment(int how) – To **set the alignment of text** in a label.

## Example to creating two labels to display text

```
import java.awt.*;
class LabelDemo1
{
	public static void main(String args[])
	{
		Frame  f= new Frame("Label Demo");
		Label lab1=new Label("Welcome to AWT 
programming");
		lab1.setBounds(50,50,200,30);
		Label lab2=new Label("This is one of the best 
feature Java");
		lab2.setBounds(50,100,200,30);
		f.add(lab1);
		f.add(lab2);
		f.setSize(500,500);
		f.setLayout(null);
		f.setVisible(true);
	}
}
```


Label Demo

Welcome to AWT programming

This is one of the best feature Java

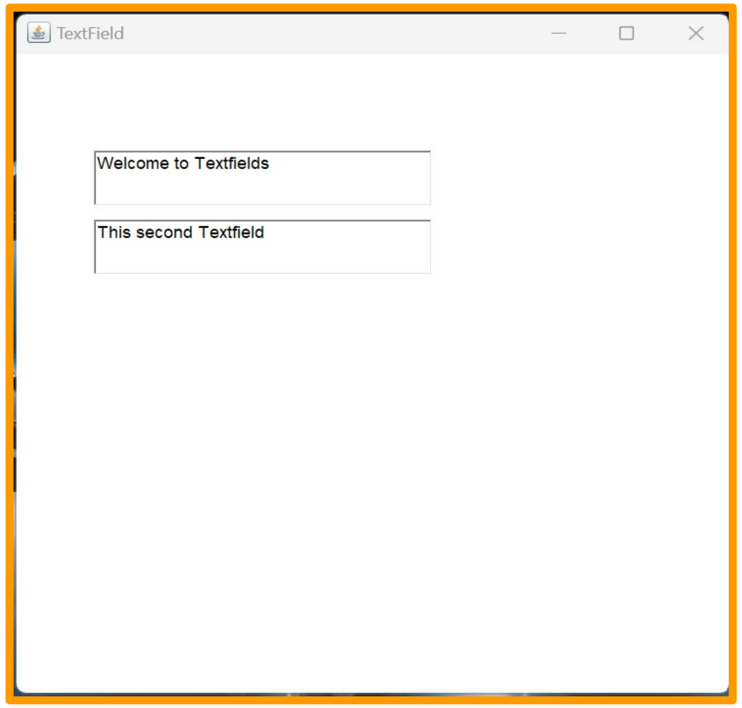
## iii.TextField

A text field or **text box is a single line text entry control** which **allows the user to enter a single line of text.**

- A **text field can be created** using the TextField class along with **its following constructors**:
  - ✓ TextField ( )
  - ✓ TextField (int numChars)
  - ✓ TextField (String str)
  - ✓ TextField (String str, int numChars)
- In the above constructors **numChars specifies** the **width of the text field**, and **str specifies the initial text** in the text field.

# Example to creating two TextFields

```
import java.awt.*;
class TextFieldDemo1
{
     public static void main(String args[])
     {
          Frame f= new Frame("TextField");
          TextField t1=new TextField("Welcome to Textfields");
          t1.setBounds(60,100, 230,40);
          TextField t2=new TextField("This second Textfield");
          t2.setBounds(60,150, 230,40);
          f.add(t1);
          f.add(t2);
          f.setSize(500,500);
          f.setLayout(null);
          f.setVisible(true);
     }
}
```


TextField

Welcome to Textfields

This second Textfield

## iv.TextArea

- It is used for **displaying multiple-line text.**
- **A text area** is a multi-line text entry control in which **user can enter multiple lines of text.**
- A text area can be created using the following constructors:
  - ✓ TextArea ()
  - ✓ TextArea (int numLines, int numChars)
  - ✓ TextArea (String str)
  - ✓ TextArea (String str, int numLines, int numChars)
  - ✓ TextArea (String str, int numLines, int numChars, int sBars)
- In the above constructors,
  - **numLines specifies the height of the text area**,
  - **numChars specifies** the **width of the text area,**
  - **str specifies** the **initial text** in the text area
  - **sBars specifies** the **scroll bars.**

## iv.TextArea

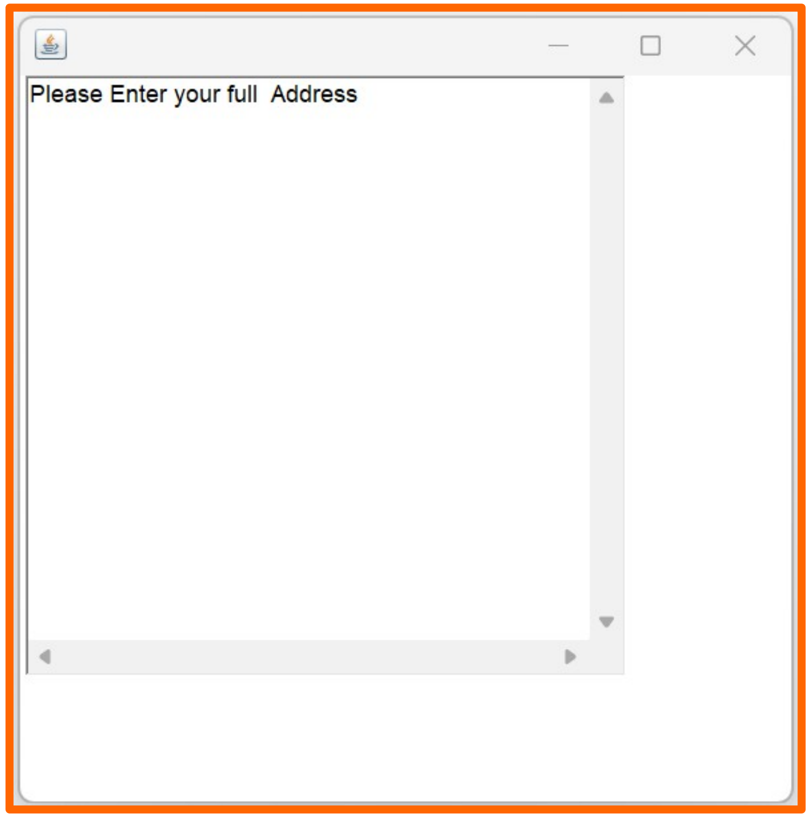
- ✓ SCROLLBARS_BOTH
- ✓ SCROLLBARS_NONE
- ✓ SCROLLBARS_HORIZONTAL_ONLY
- ✓ SCROLLBARS_VERTICAL_ONLY

- Following are **some of the methods available** in the **TextArea class:**

1. **void setText(String str)** – To assign or set the text in a text area.
2. **void select(int startindex, int endindex)** – To select the text in text field from startindex to endindex – 1.
3. **void insert(String str, int index)** – To insert the given string at the specified index.
4. **void replaceRange(String str, int startIndex, int endIndex)** – To replace the text from startIndex to endIndex – 1 with the given string.

## Example to creating TextArea

```
import java.awt.*;
public class TAExample
{
    TAExample()
    {

        Frame f = new Frame();
        TextArea ta= new TextArea("Please Enter
your full  Address");
        ta.setBounds(10, 30, 300, 300);
        f.add(ta);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{
  TAExample te=new TAExample();
}
}
```



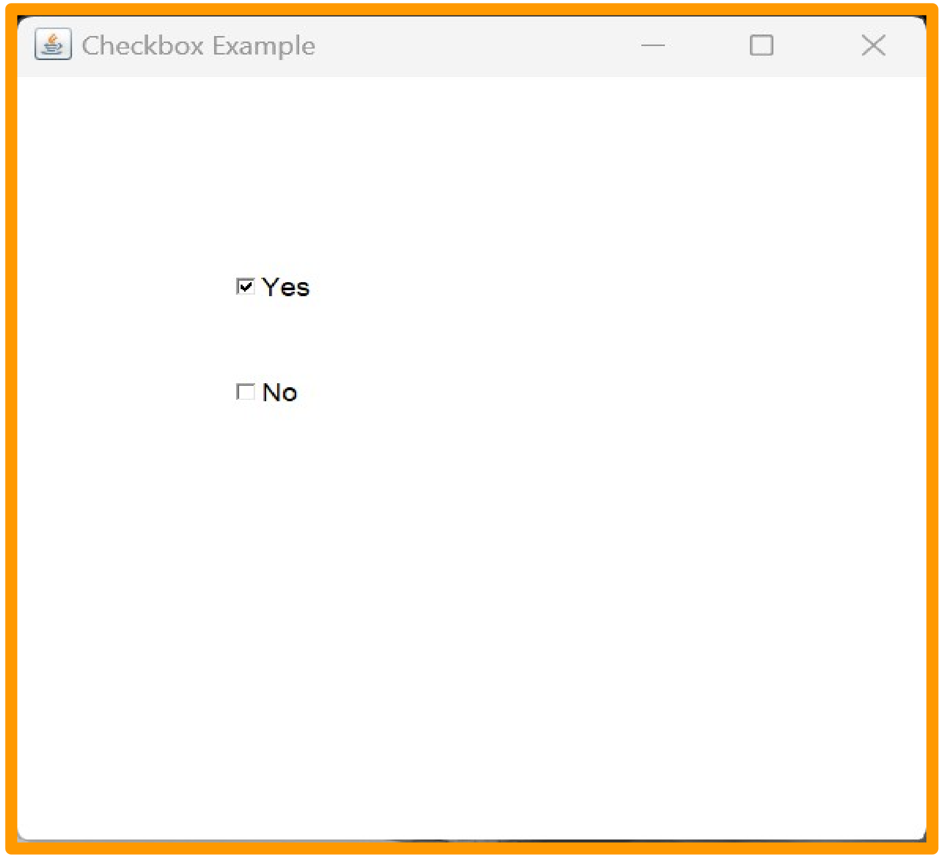
```
Please Enter your full  Address
```

## v.Checkbox

- It is **used when we want** to **select only one option** i.e true or false.
- When the **checkbox is checked then its state is "on" (true)** else it is "off"(false).
- A **checkbox** control can be **created using following constructors:**
    - Checkbox()
    - Checkbox(String str)
    - Checkbox(String str, boolean b)
- Following are various methods available in the Checkbox class:
    - **boolean getState()** – To retrieve the state of a checkbox.
    - **void setState(boolean on)** – To set the state of a checkbox.
    - **String getLabel()** – To retrieve the text of a checkbox.
    - **void setLabel(String str)** – To set the text of a checkbox.

## Example to creating checkbox

```
import java.awt.*;
public class CheckboxDemo1
{
       CheckboxDemo1()
      {
            Frame  f= new Frame("Checkbox Example");
            Checkbox c1 = new Checkbox("Yes", true);
            c1.setBounds(100,100, 60,60);
            Checkbox c2 = new Checkbox("No");
            c2.setBounds(100,150, 60,60);
            f.add(c1);
            f.add(c2);
            f.setSize(400,400);
            f.setLayout(null);
           f.setVisible(true);
      }
      public static void main(String args[])
       {
            CheckboxDemo1  c=new CheckboxDemo1();
       }
}
```


Checkbox Example

[✓] Yes

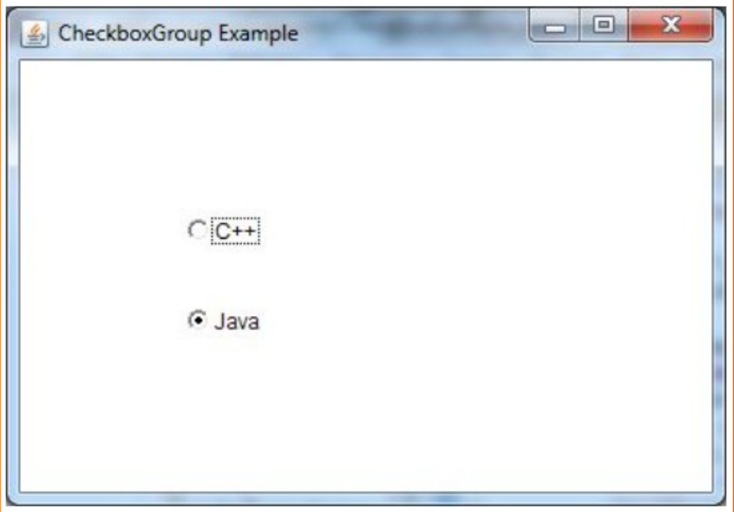
[ ] No

## Vi. CheckboxGroup

- It is used to **group a set of Checkbox.**
- When **Checkboxes are grouped** then **only one box can be checked at a time.**
- In AWT, there is **no separate class for creating radio buttons.**
- The **difference between a checkbox** and **radio button** is, a **user can select one or more checkboxes**. Whereas, a **user can select only one radio button in a group.**
- **Radio buttons** can be **create by CheckboxGroup.**

# Example for creating CheckboxGroup

```
import java.awt.*;
public class CheckboxGroupExample
{
        CheckboxGroupExample()
    {
        Frame f= new Frame("CheckboxGroup Example");
        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox c1 = new Checkbox("C++", cbg, false);
        c1.setBounds(100,100, 50,50);
        Checkbox c2 = new Checkbox("Java", cbg, true);
        c2.setBounds(100,150, 50,50);
        f.add(c1);
        f.add(c2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
{
  CheckboxGroupExample c=new CheckboxGroupExample();
}
}
```

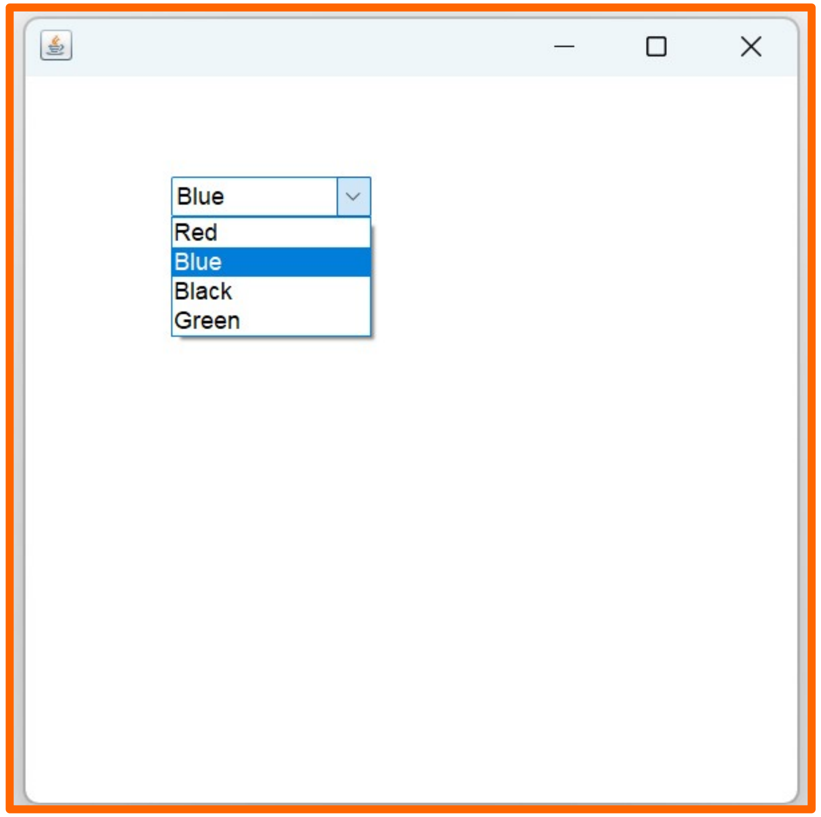


## Vii.ChoiceboxGroup

- It is **used for creating** a **drop-down** menu of choices.
- When a **user selects a particular item** from the **drop-down then it is shown** on **the top of the menu.**
- When a **user clicks on a drop down box**, it **pops up a list of items** from which **user can select a single item.**
- Following are various methods available in Choice class:
    1. **void add(String name)** – To add an item to the drop down list.
    2. **int getItemCount()** – To retrieve the number of items in the drop down list.

# Example for creating choiceBox

```
import java.awt.*;
public class ChoiceDemo
{
      ChoiceDemo()
     {
           Frame f= new Frame();
           Choice c=new Choice();
           c.setBounds(80,80, 100,100);
           c.add("Red");
           c.add("Blue");
           c.add("Black");
           c.add("Green");
           f.add(c);
           f.setSize(400,400);
           f.setLayout(null);
           f.setVisible(true);
     }
     public static void main(String args[])
     {
            ChoiceDemo c1= new ChoiceDemo();
     }
}
```



# Java LayoutManagers

- The **LayoutManagers** are **used** to **arrange components** in a **particular manner.**
- It facilitates us **to control the positioning** and **size of the components** in **GUI** forms.
- **LayoutManager is an interface** that is **implemented** by **all the classes of layout managers.**
- There are the **four classes** that **represent** the **layout** managers:

  i. **BorderLayout**
  ii. **FlowLayout**
  iii. **GridLayout**
  iv. **CardLayout**

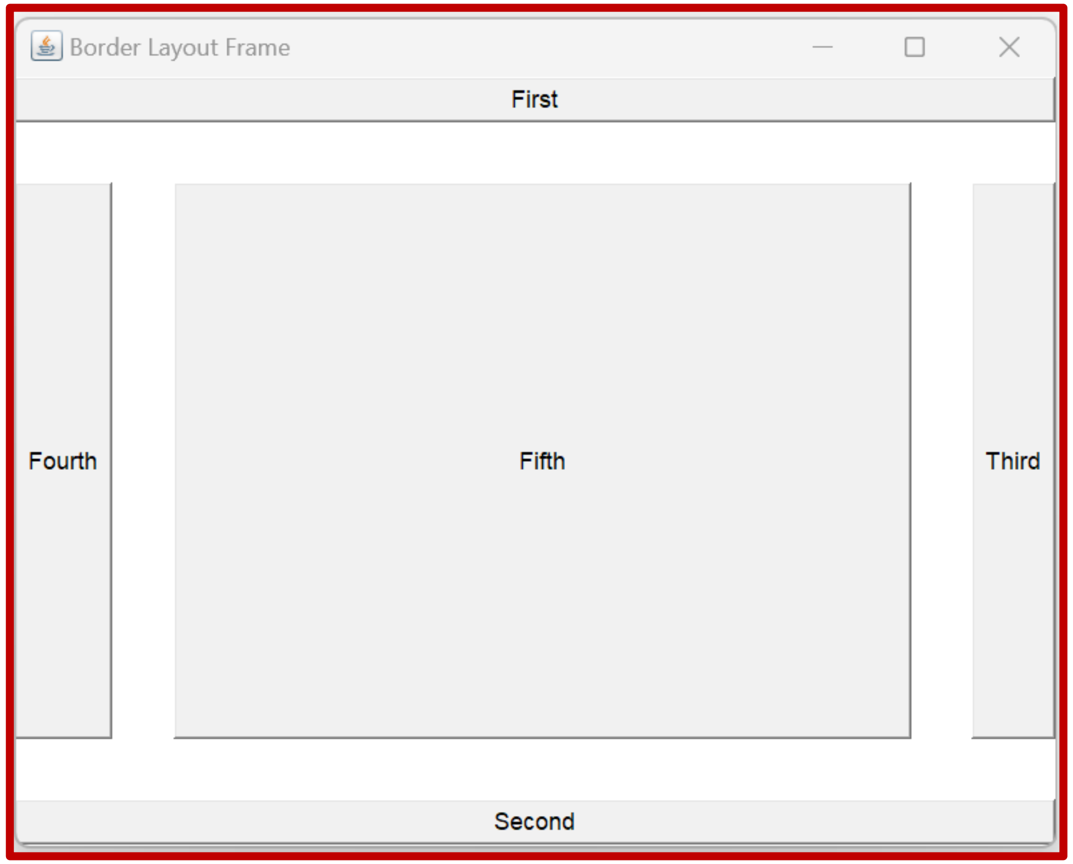
# i. Border Layout

- It is **used to arrange the components** in **five regions**: **north, south, east, west, and center.**
- **Each region** (area) may **contain one component only.**
- It is the **default layout of a frame** or **window.**
- The **BorderLayout provides five constants** for each **region:**
    - BorderLayout.NORTH
    - BorderLayout.SOUTH
    - BorderLayout. EAST
    - BorderLayout. WEST
    - BorderLayout. CENTER

| SNO | Method Name | Purpose |
|---|---|---|
| 1 | **BorderLayout()** | **creates a border layout** but **with no gaps between the components.** |
| 2 | **BorderLayout(int hgap,** | **creates a border layout** with the **given horizontal and** |

## Border Layout Example Program

```
import java.awt.*;
public class BLExample
{
  public static void main(String[] args)
  {
   Frame f= new Frame("Border Layout
Frame");
   Button b1= new Button("First");
   Button b2=new Button("Second");
   Button b3=new Button("Third");
   Button b4=new Button("Fourth");
   Button b5=new Button("Fifth");
   f.setLayout(new
BorderLayout(30,30));
   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);
   f.setSize(300,300);
   f.setVisible(true);
  }
}
```



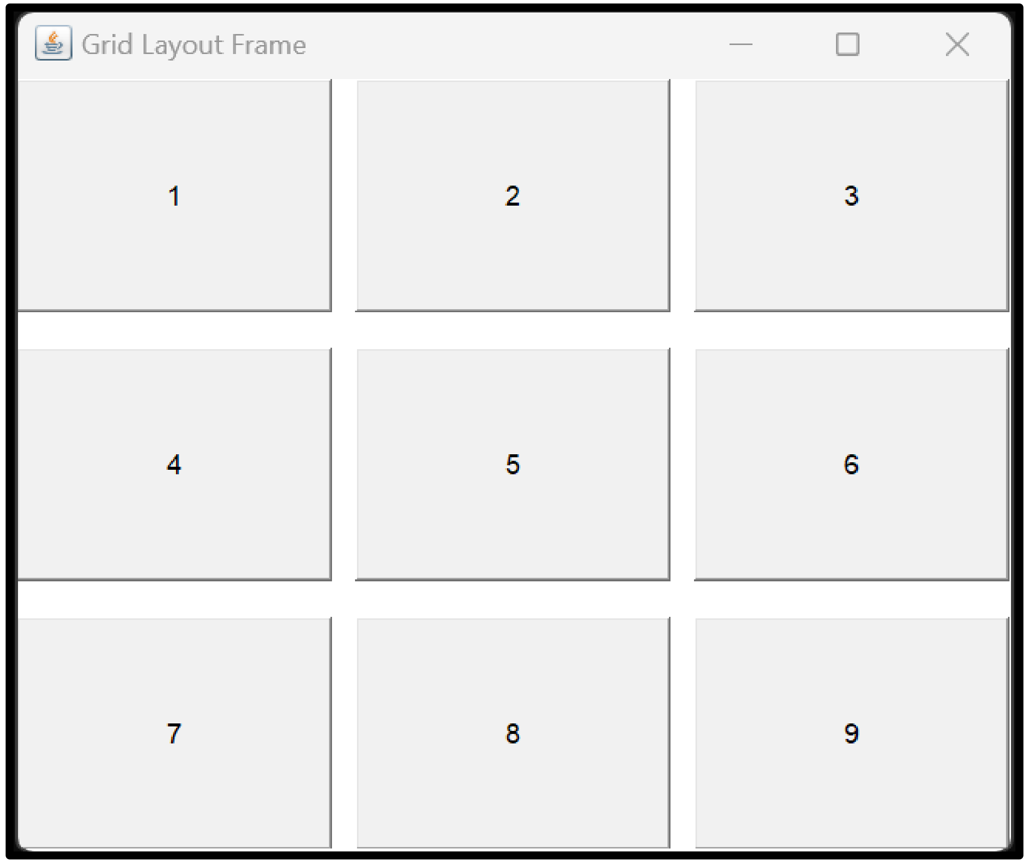
# ii. Grid Layout

- It is **used to arrange** the **components** in a **rectangular grid.**
- **One component** is displayed in **each rectangle.**
- **You start at row one**, **column one**, then **move across** the **row until it's full**, then **continue on to the next row.**
- It is **widely used for arranging components** in **rows** and **columns.**
- The **order of placement of components is directly dependant** on the **order in which they are added** to the **frame or panel.**
- **Constructors** of GridLayout class:

| SNO | Method Name | Purpose |
|---|---|---|
| 1 | **GridLayout()** | **creates a grid layout** with **one column per component** in a row. |
| 2 | **GridLayout(int rows, int columns)** | **creates a grid layout** with the **given rows** and **columns** but **no gaps** between the components. |
|  | [illegible] | **creates a grid layout** with the **given** |

# Grid Layout Example Program

```
import java.awt.*;
public class GLExample
{
  public static void main(String[] args)
  {
   Frame f= new Frame("Grid Layout Frame");
   Button b1= new Button("1");
   Button b2=new Button("2");
   Button b3=new Button("3");
   Button b4=new Button("4");
   Button b5=new Button("5");
   Button b6=new Button("6");
   Button b7=new Button("7");
   Button b8=new Button("8");
   Button b9=new Button("9");
   f.setLayout(new
GridLayout(3,3,10,15));
   f.add(b1);
   f.add(b2);
   f.add(b3);
   f.add(b4);
   f.add(b5);
   f.add(b6);
   f.add(b7);
   f.add(b8);
   f.add(b9);
   f.setSize(300,300);
   f.setVisible(true);
  }
}
```



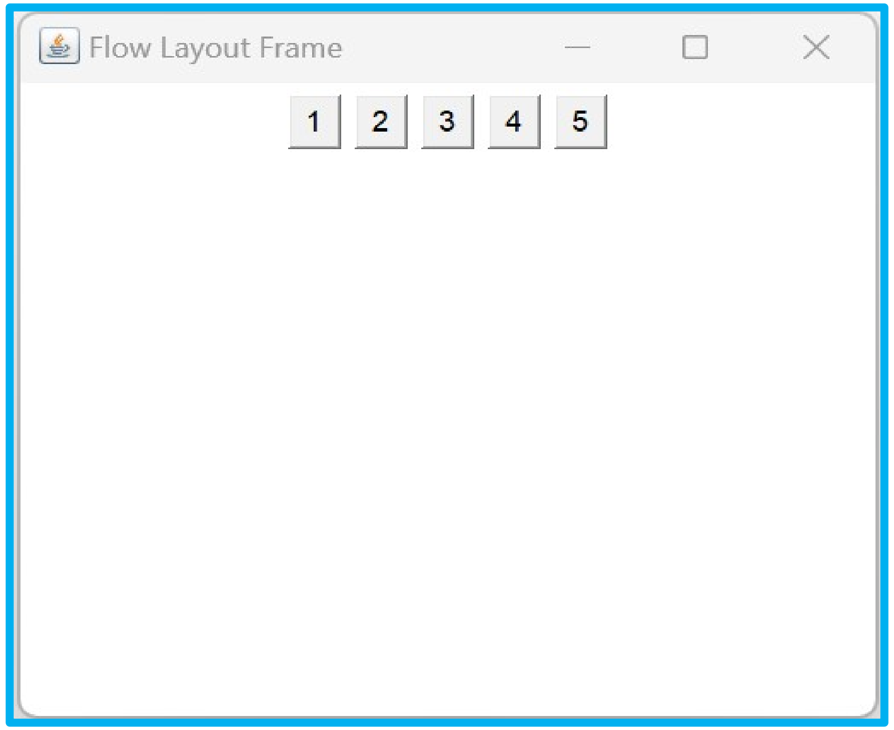
# iii. FlowLayout

- **FlowLayout** class is **used to arrange** the **components in a line, one after another** (in a flow).
- It is the **default layout of the applet** or **panel.**
- **It is basically helps develop more responsive UI** and keep the components in a **free flowing manner.**
- [illegible]
- **Constructors** of FlowLayout class:

| SNO | Method Name | Purpose |
|---|---|---|
| 1 | **FlowLayout()** | **creates a flow layout** with **centered alignment** and a **default 5 unit horizontal** and **vertical gap.** |
| 2 | **FlowLayout(int align)** | **creates a flow layout** with the **given alignment** and a default 5 unit horizontal and vertical gap. |
| 3 | **FlowLayout(int align, int hgap, int vgap)** | **creates a flow layout** with the **given alignment** and the **given horizontal and vertical gap.** |

# Flow Layout Example Program

```
import java.awt.*;
public class FLExample
{
  public static void main(String[] args)
  {
   Frame f= new Frame("Flow Layout
Frame");
   Button b1= new Button("1");
   Button b2=new Button("2");
   Button b3=new Button("3");
   Button b4=new Button("4");
   Button b5=new Button("5");
   f.setLayout(new FlowLayout());
   f.add(b1);
   f.add(b2);
   f.add(b3);
   f.add(b4);
   f.add(b5);
   f.setSize(300,300);
   f.setVisible(true);
  }
}
```



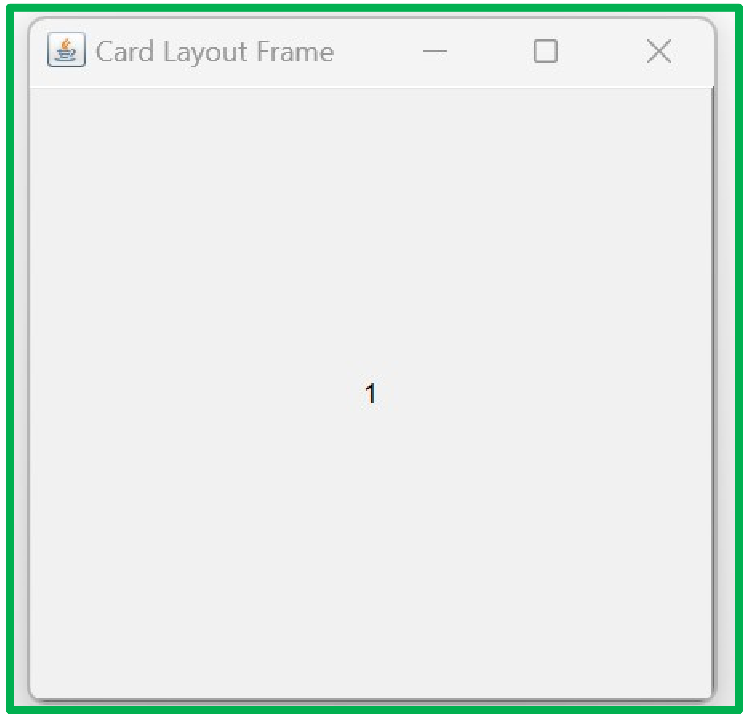
# CardLayout

- **CardLayout class manages** the **components in such a manner** that **only one component is visible** at a time and **hiding the rest.**
- It **treats each component as a card.**
- It is **rarely used** and is **utilized to stack up components one above another.**
- **Constructors** of CardLayout Class

| SNO | Method Name | Purpose |
|---|---|---|
| 1 | **CardLayout()** | **creates a card layout** with **zero horizontal and vertical gap.** |
| 2 | **CardLayout(int hgap, int vgap)** | **creates a card layout** with the **given horizontal** and **vertical gap.** |

## Card Layout Example Program

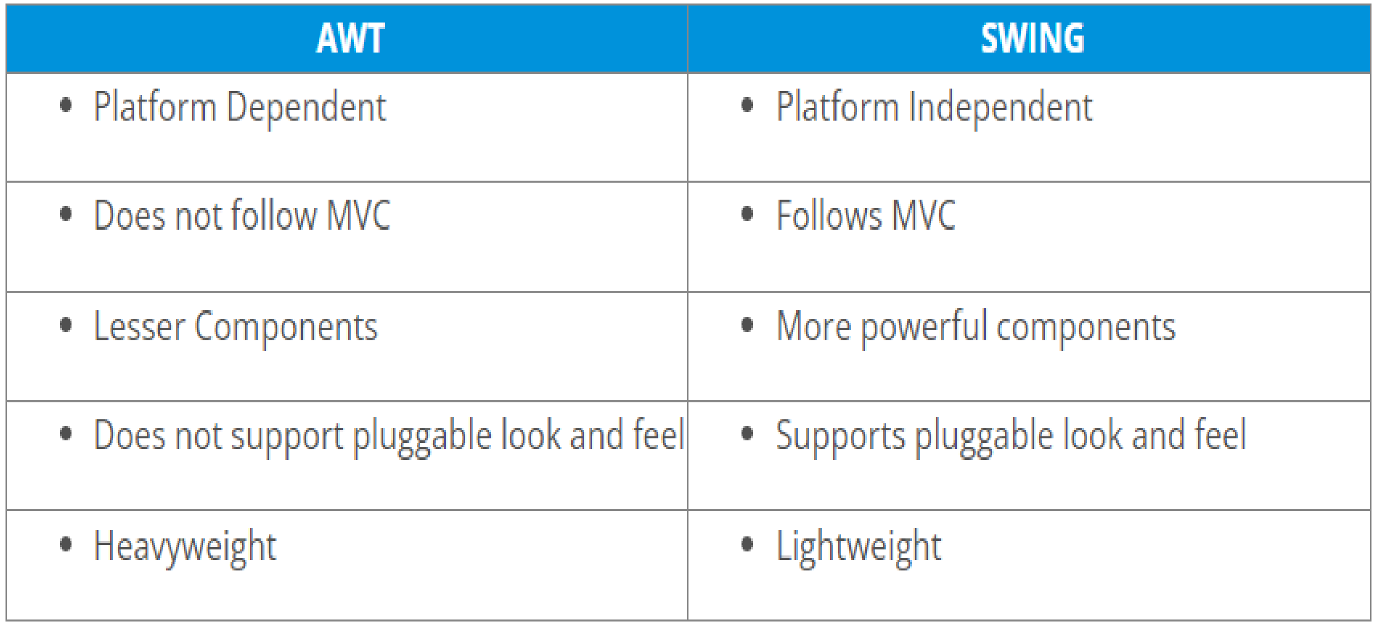
```
import java.awt.*;
public class CLExample
{
  public static void main(String[] args)

  {
   Frame f= new Frame("Card Layout
Frame");
   Button b1= new Button("1");
   Button b2=new Button("2");
   Button b3=new Button("3");
   Button b4=new Button("4");
   Button b5=new Button("5");
   f.setLayout(new CardLayout());
   f.add(b1);
   f.add(b2);
   f.add(b3);
   f.add(b4);
   f.add(b5);
   f.setSize(300,300);
   f.setVisible(true);
  }
```



# Swings

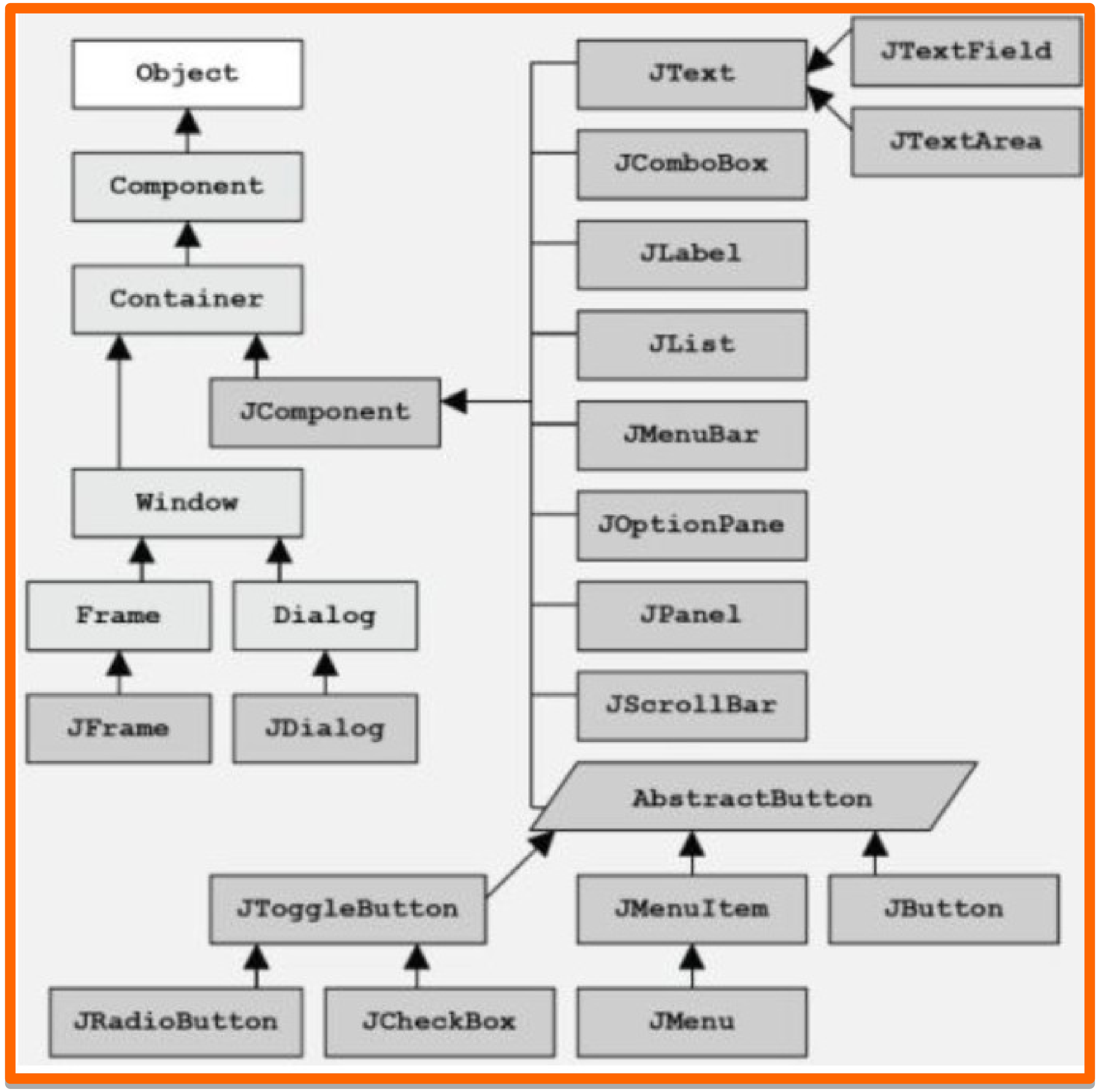
- Swing in Java is a **lightweight GUI toolkit**
- **Swing** in Java is a **Graphical User Interface (GUI) toolkit** that **includes the GUI components.**
- **Swing provides a rich set packages** to make **sophisticated GUI components** for Java applications.
- **Swing** is a **part of Java Foundation Classes(JFC),** which is **an API for Java GUI.**

## Difference Between AWT and Swing



| AWT | SWING |
|---|---|
| • Platform Dependent | • Platform Independent |
| • Does not follow MVC | • Follows MVC |
| • Lesser Components | • More powerful components |
| • Does not support pluggable look and feel | • Supports pluggable look and feel |
| • Heavyweight | • Lightweight |

# Swings


Object
Component
Container
JComponent
Window
Frame
Dialog
JFrame
JDialog
JText
JTextField
JTextArea
JComboBox
JLabel
JList
JMenuBar
JOptionPane
JPanel
JScrollBar
AbstractButton
JToggleButton
JMenuItem
JButton
JRadioButton
JCheckBox
JMenu

## Swings

**JFrame:**

- JFrame is a **top-level container** that **represents the main window** of a **GUI** application. It **provides a title bar, and minimizes, maximizes, and closes buttons.**

**JPanel:**

- JPanel is a **container** that can **hold other components**. It is **commonly used** to **group related components together.**

**JButton:**

- JButton is a **component** that **represents** a **clickable button**. It is **commonly used to trigger actions** in a **GUI** application.

**JLabel:**

- JLabel is a **component** that **displays text** or **an image.** It is commonly used to **provide information** or to label **other components.**

# Swings

## JTextField:

- JTextField is a component that **allows the user to input text**. It is commonly **used to get input** from the user, such as a name or an address.

## JCheckBox:

- JCheckBox is a component that represents a checkbox. **It is commonly used to get a binary input** from the user, **such as whether or not to enable** a feature.

## JList:

- JList is a component that **represents a list of elements.** It is typically **used to display a list of options** from which the **user can select one or more items.**

## JTable:

- JTable is a component that **represents a data table**. It is **typically used to present data in a tabular fashion**, such as a list of products or a list of orders.

# Swings

**JScrollPane:**

- ✓ JScrollPane is a **component** that provides **scrolling functionality** to **other components**.
- ✓ It is **commonly used** to **add scrolling to a panel or a table.**

## Creating a JFrame

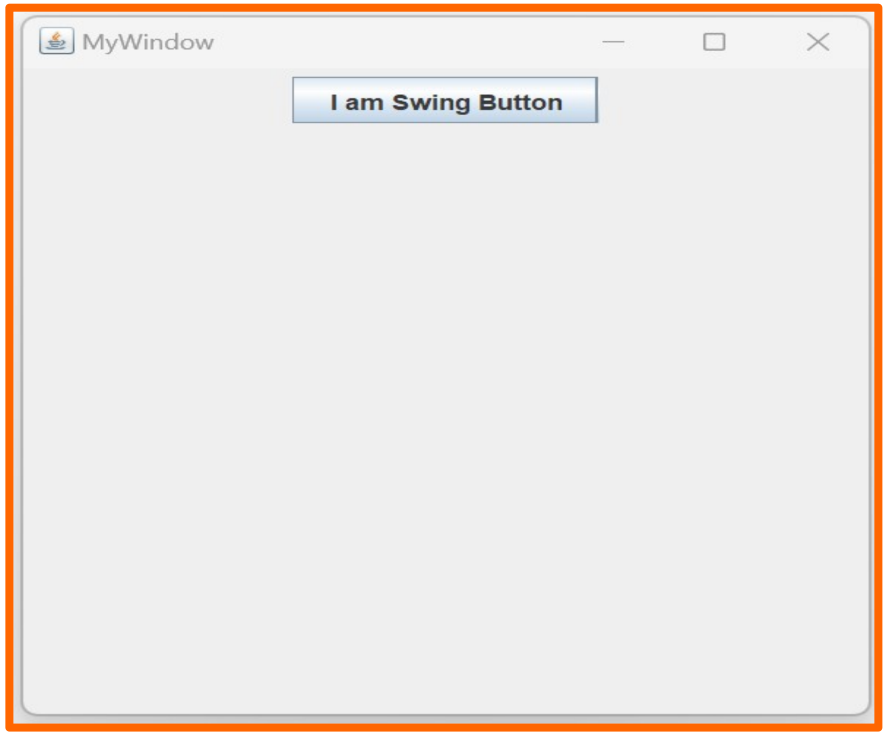
- There are two ways to create a JFrame Window.

  **i. By instantiating JFrame class.**

  **ii. By extending JFrame class.**

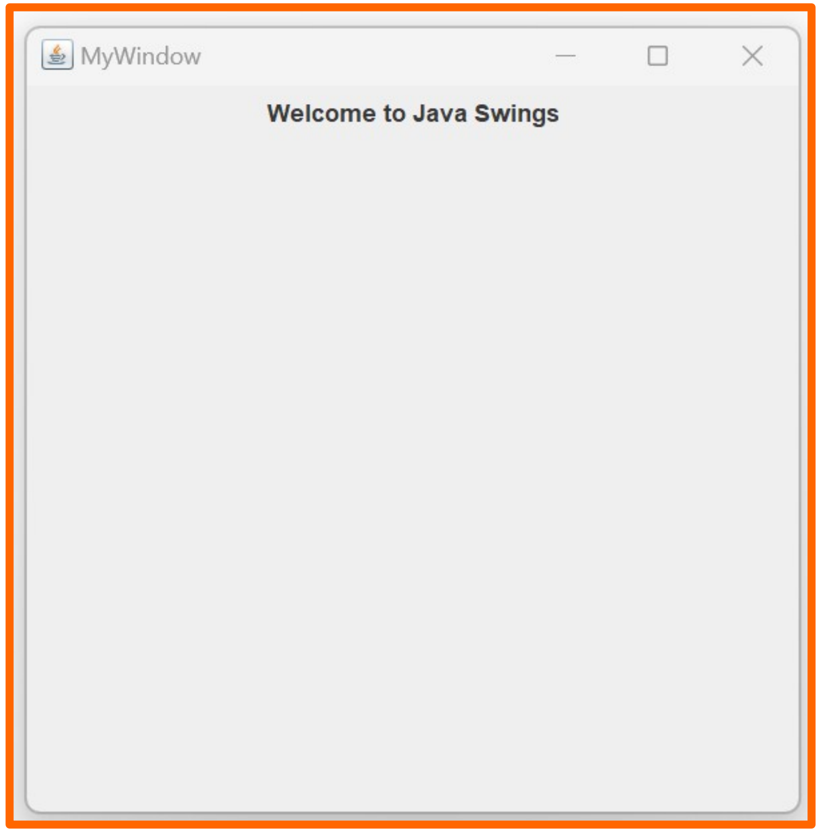
## Creating JFrame Window by instantiating JFrame class

```
//Creating JFrame window by instantiating JFrame
class
import javax.swing.*;
import java.awt.*;
public class First
{
      First( )
       {
       JFrame jf = new JFrame("MyWindow");

      JButton btn = new JButton("I am Swing
Button");
      jf.add(btn);

      jf.setLayout(new FlowLayout());


jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS
E);
      jf.setSize(400, 400);

      jf.setVisible(true);

      }
      public static void main(String[] args)
      {
```



## Creating JFrame Window by Extending JFrame class

```
//Creating JFrame window by extending JFrame class
import javax.swing.*; //importing swing package
import java.awt.*; //importing awt package
public class Second extends JFrame
{
	Second()
	{
		setTitle("MyWindow");
		JLabel lb = new JLabel("Welcome to Java 
Swings");
		add(lb);
		setLayout(new FlowLayout());
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(400, 400);
		setVisible(true);
	}

	public static void main(String[] args)
	{
		Second sf=new Second();
	}
}
```

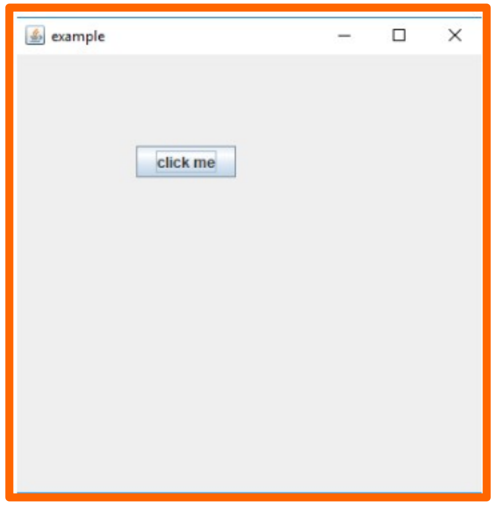


Welcome to Java Swings

# JPanel Class

It inherits the JComponent class and **provides space for an application** which **can attach any other component**

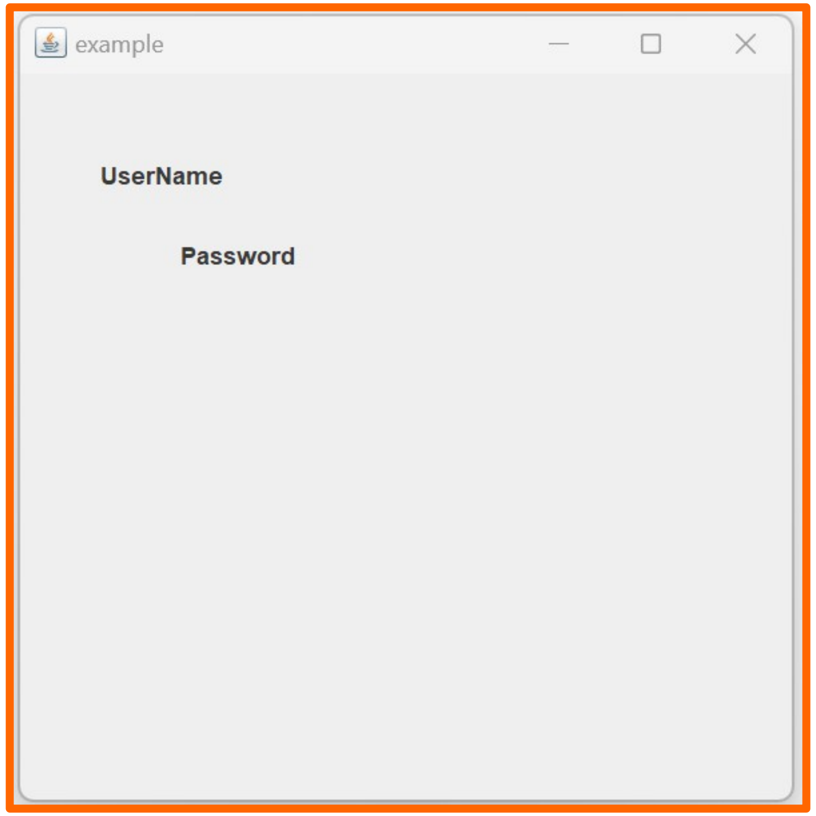
```
import java.awt.*;
import javax.swing.*;
public class Example
{
        Example()
        {
                JFrame jf = new JFrame("example");
                JPanel p = new JPanel();
                p.setBounds(40,70,200,200);
                JButton b = new JButton("click me");
                b.setBounds(60,50,80,40);
                p.add(b);
                jf.add(p);
                jf.setSize(400,400);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                jf.setLayout(null);
                jf.setVisible(true);
        }
        public static void main(String args[])
        {
                new Example();
        }
}
```



# JLabel

- It is used for **placing text** in a **container.**
- Only **Single line text is allowed** and the text **can not be changed**

```
import javax.swing.*;
public class JLabExample
{
	public static void main(String args[])
	{
		JFrame a = new JFrame("example");
		JLabel L1= new JLabel("UserName");
		JLabel L2= new JLabel("Password");
		L1.setBounds(40,40,90,20);
		a.add(L1);
		L2.setBounds(80,80,90,20);
		a.add(L2);
		a.setSize(400,400);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		a.setLayout(null);
		a.setVisible(true);
	}
}
```

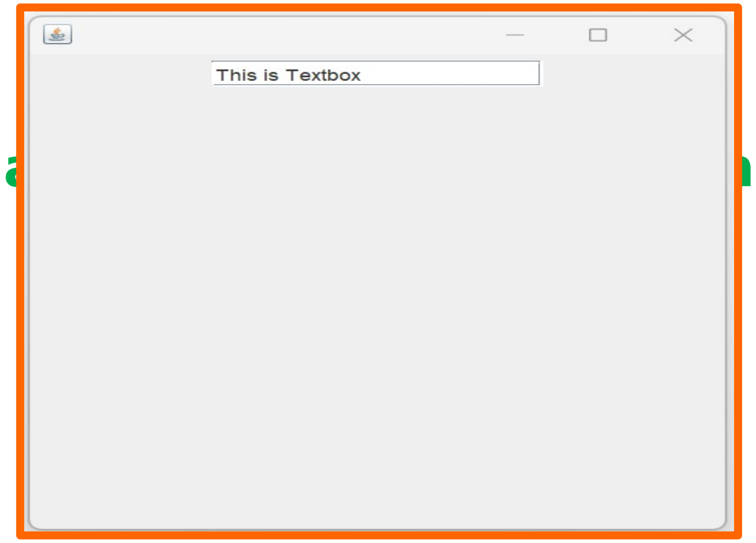


# JTextField

- **JTextField** is **used for taking input of single line of text**. It is most widely used text component.
  - ✓ **JTextField(int cols)**
  - ✓ **JTextField(String str, int cols)**
  - ✓ **JTextField(String str)**
- ***cols* represent** the **number of columns** in text field.
- It is used to **a**

**ne text.**

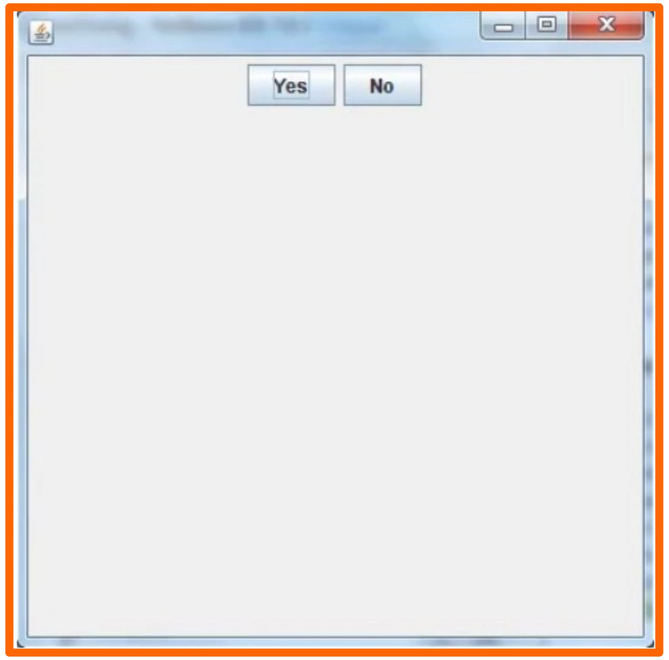
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class MyTextField
{
  public MyTextField()
  {
    JFrame jf = new JFrame();
    JTextField jtf = new JTextField("This is Textbox",20);
    jf.add(jtf);
    jf.setLayout(new FlowLayout());
    
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jf.setSize(400, 400);
    jf.setVisible(true);
  }
  public static void main(String[] args)
  {
    new MyTextField();
  }
}
```


This is Textbox

# JButton

JButton class **provides functionality of a button**. It is **used to create button component**

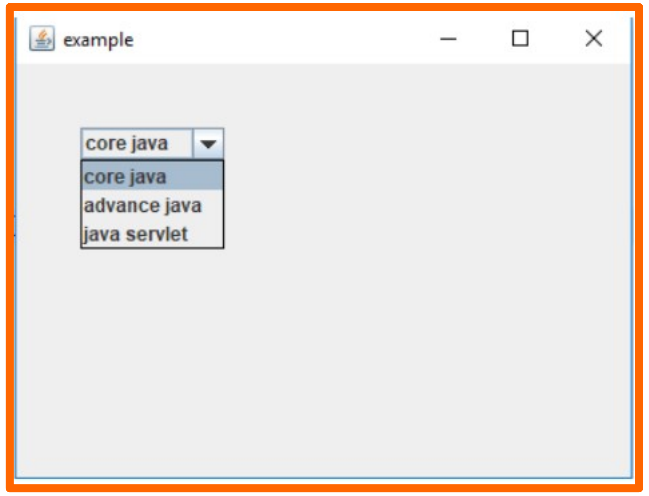
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class testswing
{

  testswing()
  {
     JFrame jf = new JFrame("example");
    JButton bt1 = new JButton("Yes");
    JButton bt2 = new JButton("No");
    jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
    jf.setLayout(new FlowLayout());
    jf.setSize(400, 400);
    jf.add(bt1);
    jf.add(bt2);
    setVisible(true);
  }
  public static void main(String[] args)
  {
    testswing ts=new testswing();
  }
}
```



# JComboBox

- It inherits the JComponent class and is **used to show pop up menu of choices**.
- Combo box is a combination of text fields and drop-down list.

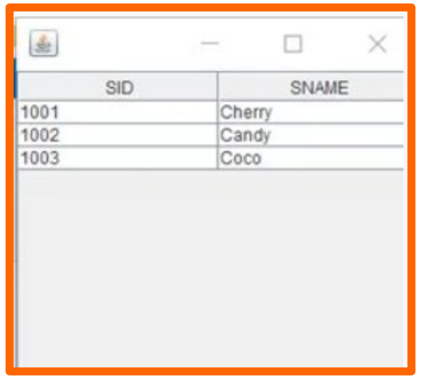
```
import javax.swing.*;
public class Example
{
        Example()
        {
                JFrame a = new JFrame("example");
                string courses[] = { "core java","advance java", "java
servlet"};
                JComboBox c = new JComboBox(courses);
                c.setBounds(40,40,90,20);
                a.add(c);
                a.setSize(400,400);
                a.setLayout(null);
                a.setVisible(true);
        }
        public static void main(String args[])
        {
                Example  e=new Example();
        }
}
```



# JTable

- It **used to draw a table to display data.**
- The **JTableContains 2 constructors**:
  i. JTable()
  ii. JTable(Object[][] rows, Object[] columns)

```
import javax.swing.*;
public class STableDemo1
{
	STableDemo1()
	{
	JFrame  jf=new JFrame();
	String table_data[][]={ {"1001","Cherry"}, {"1002","Candy"},
{"1003","Coco"}};
	String table_column[]={"SID","SNAME"};
	JTable jt=new JTable(table_data,table_column);
	jt.setBounds(30,40,200,300);
	JScrollPane tsp=new JScrollPane(jt);
	jf.add(tsp);
	jf.setSize(300,400);
	jf.setVisible(true);
	}
	public static void main(String[] args)
	{
	new STableDemo1();
	}
}
```

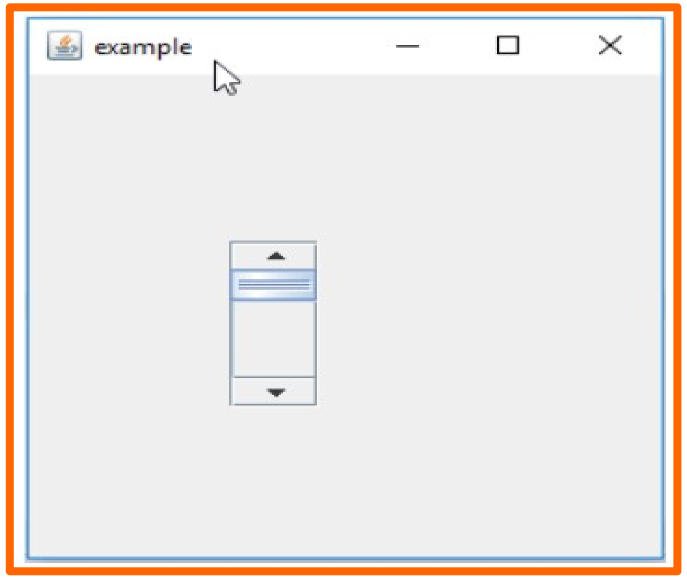


| SID | SNAME |
|---|---|
| 1001 | Cherry |
| 1002 | Candy |
| 1003 | Coco |

# JScrollBar

- It is used to **add scroll bar, both horizontal and vertical.**

```
import javax.swing.*;
class Example
{
	Example()
	{
		JFrame a = new JFrame("example");
		JScrollBar b = new JScrollBar();
		b.setBounds(90,90,40,90);
		a.add(b);
		a.setSize(300,300);
		a.setLayout(null);
		a.setVisible(true);
	}
	public static void main(String args[])
	{
		Example e=new Example();
	}
}
```

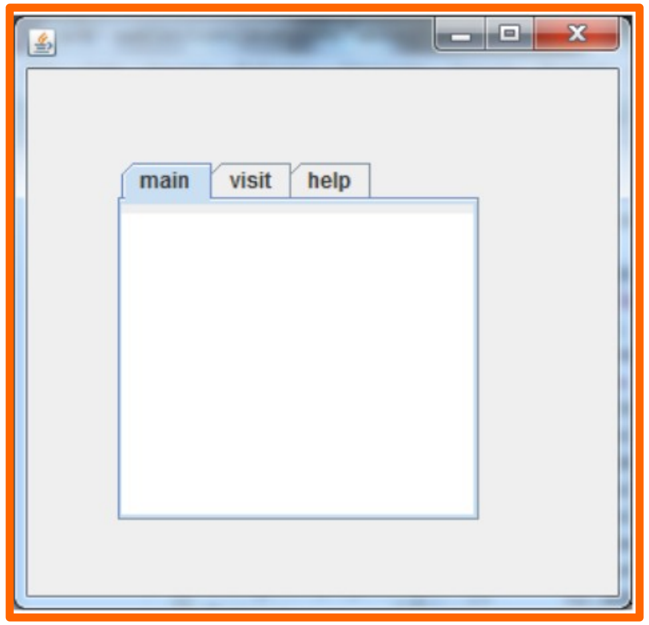


# JTabbedPane

- It is **used to switch between a group of components** by **clicking on a tab** with a **given title or icon.**

```
import javax.swing.*;
public class TabbedPaneExample
{
TabbedPaneExample()
{
    JFrame f=new JFrame();
    JTextArea ta=new
JTextArea(200,200);
    JPanel p1=new JPanel();
    p1.add(ta);
    JPanel p2=new JPanel();
    JPanel p3=new JPanel();
    JTabbedPane tp=new
JTabbedPane();
    tp.setBounds(50,50,200,200);
    tp.add("main",p1);
    tp.add("visit",p2);
    tp.add("help",p3);
    f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
```

```
public static void main(String[]
args)
{
    new TabbedPaneExample();
}
}
```

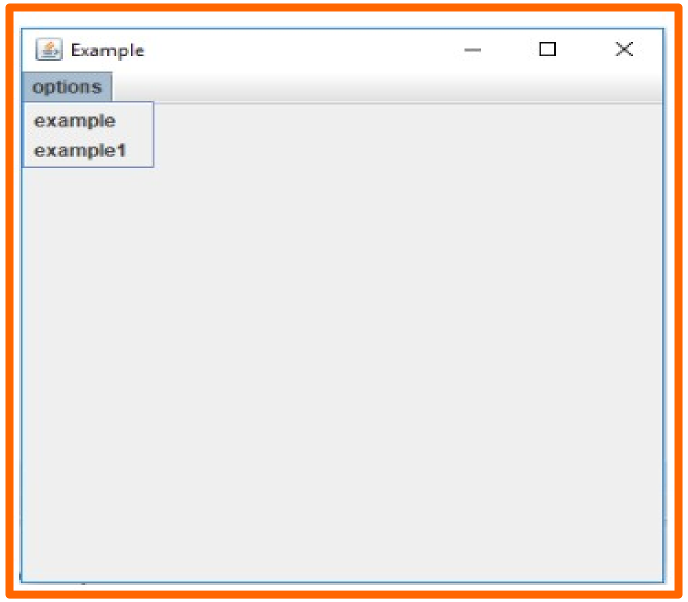


# JMenu

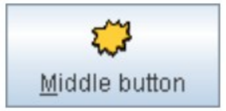
- It inherits the JMenuItem class, and is a **pull down menu component** which is **displayed from the menu bar**

```
import javax.swing.*;
class Example
{

	Example()
	{
		JFrame a = new JFrame("Example");
		JMenu m = new JMenu("options");
		JMenuBar mb = new JMenuBar();
		JMenuItem a1 = new 
JMenuItem("example");
		JMenuItem a2 = new 
JMenuItem("example1");
		m.add(a1);
		m.add(a2);
		mb.add(m);
		a.setJMenuBar(mb);
		a.setSize(400,400);
		a.setLayout(null);
		a.setVisible(true);
	}
```

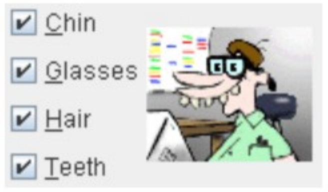
```
	public static void main(String args[])
	{
		new Example();
	}
}
```



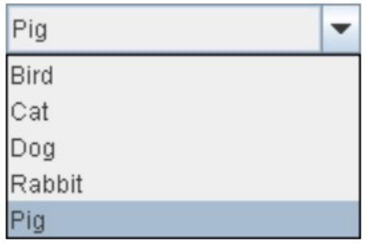
# A Visual Guide to Swing Components



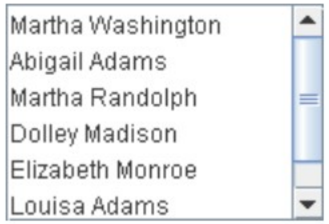
JButton



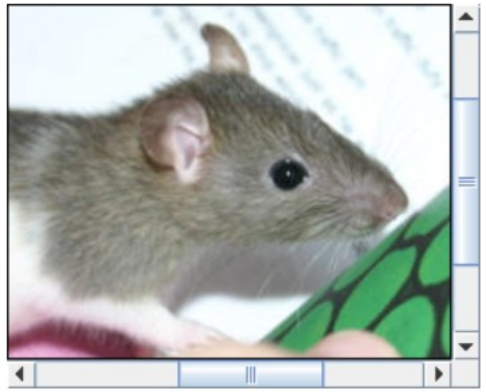
JCheckBox



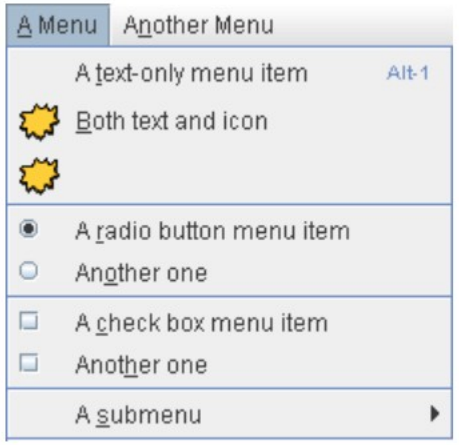
JComboBox

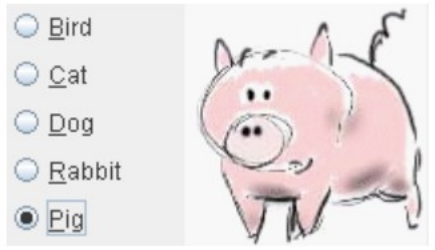


JList

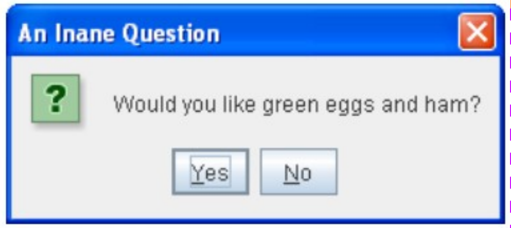


JScrollPane





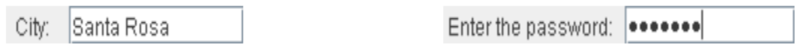
JMenu



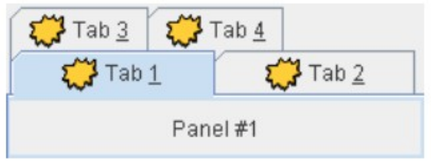
An Inane Question

Would you like green eggs and ham?

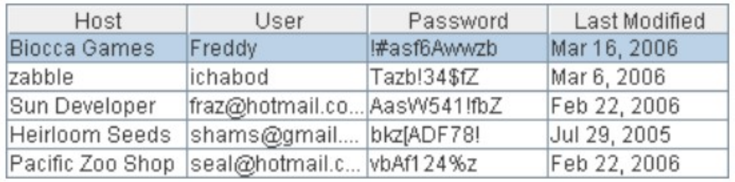
JDialog



JTextField JPasswordField

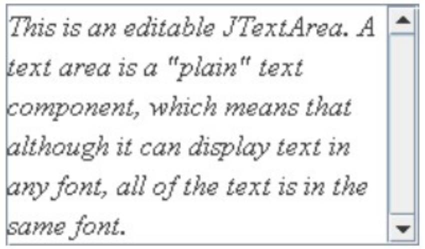


JTabbedPane



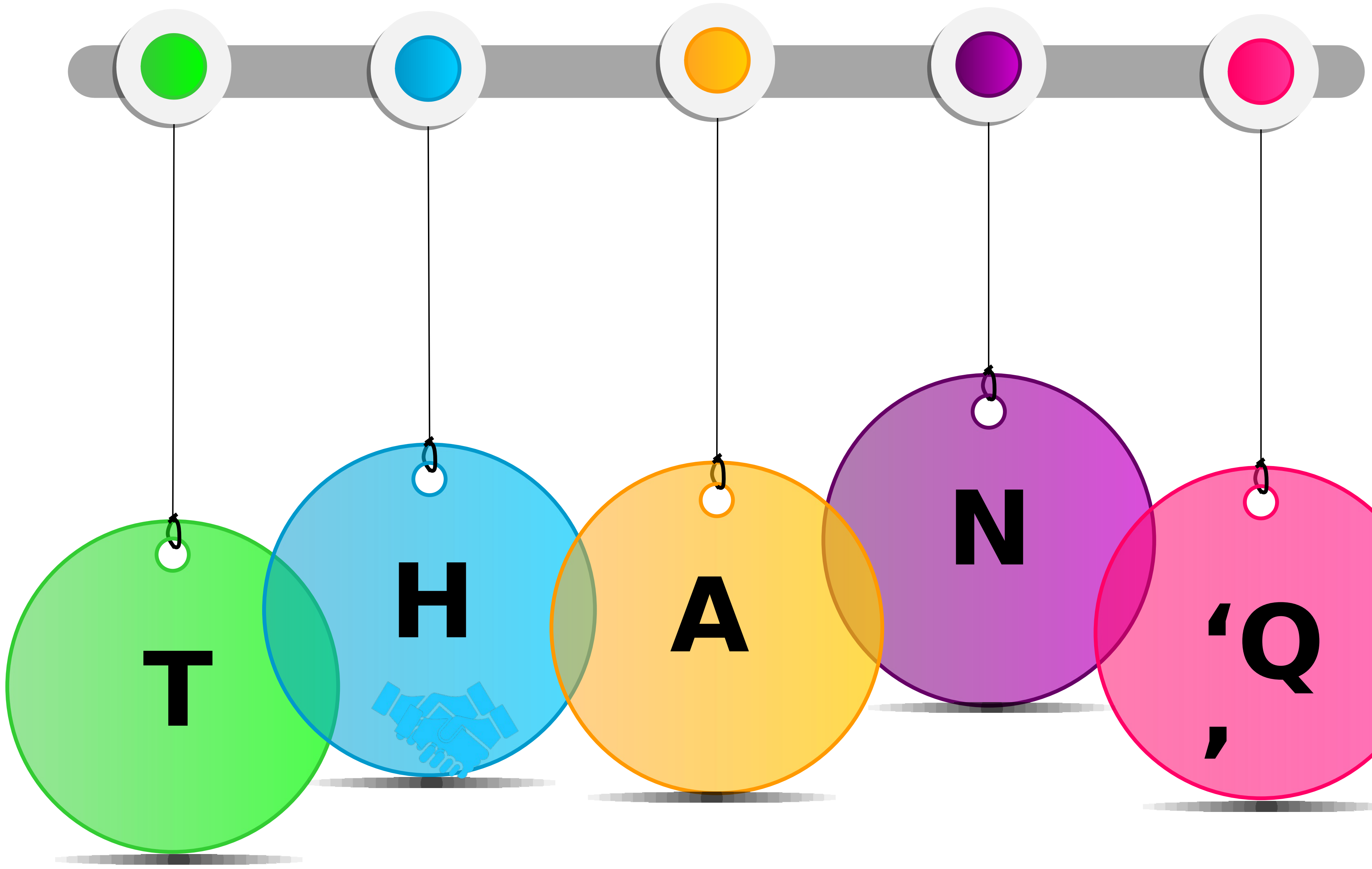
| Host | User | Password | Last Modified |
|---|---|---|---|
| Biocca Games | Freddy | !#asf6Awwzb | Mar 16, 2006 |
| zabble | ichabod | Tazb!34$fZ | Mar 6, 2006 |
| Sun Developer | fraz@hotmail.co... | AasW541!fbZ | Feb 22, 2006 |
| Heirloom Seeds | shams@gmail.... | bkz[ADF78! | Jul 29, 2005 |
| Pacific Zoo Shop | seal@hotmail.c... | vbAf124%z | Feb 22, 2006 |

JTable



*This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.*

JTextArea


T
H
A
N
‘Q
,

We wish you
all the best
2023
Success
Win
Achieve
Goals
2024
Happiness
Satisfaction
Friendship
Joy
Money
Good
Luck
Health
Peace
Love
Fun
Blessing
Happy New Year